O'REILLY®

Facilitating Software Architecture

Empowering Teams to Make Architectural Decisions

Free Chapter

Andrew Harmel-Law
Foreword by Sarah Wells

Facilitating Software Architecture

Empowering Teams to Make Architectural Decisions

This excerpt contains Chapter 11. The complete book is available on the O'Reilly Online Learning Platform and through other retailers.

Andrew Harmel-Law
Foreword by Sarah Wells

O'REILLY®

Facilitating Software Architecture
by Andrew Harmel-Law

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Louise Corrigan
Development Editor: Rita Fernando
Production Editor: Clare Laylock
Copyeditor: Shannon Turlington
Proofreader: Piper Editorial Consulting, LLC
Indexer: Potomac Indexing, LLC
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

November 2024: First Edition

Revision History for the First Edition
2024-11-08: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098151867* for release details.

978-1-098-15186-7

[LSI]

Table of Contents

CHAPTER 11

Using a Technology Radar

Being aware of the technical landscape and climate that surround you is important for an effective decision process. Decisions benefit from being informed by your colleagues' and your wider organization's collective past experiences and future intentions. But such a wealth of technical information can seem difficult both to capture and to manage.

In this chapter, I'll discuss how the collective intelligence can be distilled into a set of regularly updated guidelines—the fourth alignment mechanism—and used to inform everyone's future decisions via a fourth supporting element to the advice process: a technology radar that provides a snapshot of the technologies and techniques used in your organization right now.

I'll start introducing you to the technology radar by giving you a real-life example of how it's used in Thoughtworks before going on to describe how a technology radar fits into the advice process. I'll close by discussing how feedback from using the technology radar in deciding can—and ought to—lead to updates in the radar itself.

Sense Tech Trends and Capture Guidelines

There is always a wealth of knowledge and experience in your organization about your technical landscape and climate. Sociotechnical systems are terribly complex, so it can be valuable to uncover and maintain a distilled set of guidelines that record what has been tried, what has failed, what might be worth investigating, and what has been learned across your software engineering organization.

This isn't a trivial task. Seeking out this information can open up an ocean of inputs and opinions. Capturing and presenting the collective experience of the technical landscape needs to be done in a clean and accessible way. If not, it's easy to drown in the sea of raw information.

Technology Radars Continually Collect and Share Guidance

A technology radar captures a snapshot of the technologies and techniques used in your organization as well as your collective experience with them. First created by Thoughtworks, and structured like the radars you imagine air traffic controllers sitting in front of, the technology radar is an "opinionated" view of technology trends (both current and receding) in software languages and frameworks, tools, platforms, and techniques. Its biggest strength lies in how it uses the radar metaphor to visually represent these important aspects of the current technology landscape and the climatic movements of various technological "blips" across it.

The Thoughtworks Technology Radar, as I write, is shown in Figure 11-1.

Tracking these tech-trend "blips" as aircraft would be tracked (albeit a lot slower moving), the radar allows viewers to rapidly see, for example, what is up and coming in the world of frontend frameworks (Astro, apparently), what's the current flavor of the month (I'm reliably informed it's Svelte at the time of writing), and what's beginning to fade. (Did anyone actually enjoy using AngularJS?)[1]

The beam of the Thoughtworks Radar sweeps the global tech landscape on a six-month basis. These sweeps crowdsource experiences of Thoughtworkers on client-delivery projects, and the blips that represent these projects are positioned based on how they are being used. For a blip to appear on the radar, it has to have been used in delivery on at least one client project. To be centered, a blip has to have become a Thoughtworks "sensible default," meaning it's their go-to unless specific circumstances suggest an alternative is preferable. Taken together, the collection of blips and their respective positions is a powerful summation of the organization's collective experiences and perceptions across all clients.

A technology radar is therefore a great tool for sharing guidelines about a significant number of technology trends, in a compact and consumable form, and all sourced from a vast collective.

1 By the time you read this, my next/current/old-news picks will be laughably out of date. That's the point. Things change fast in the world of tech, and no more so than in the world of web frameworks, apart from perhaps large language models.

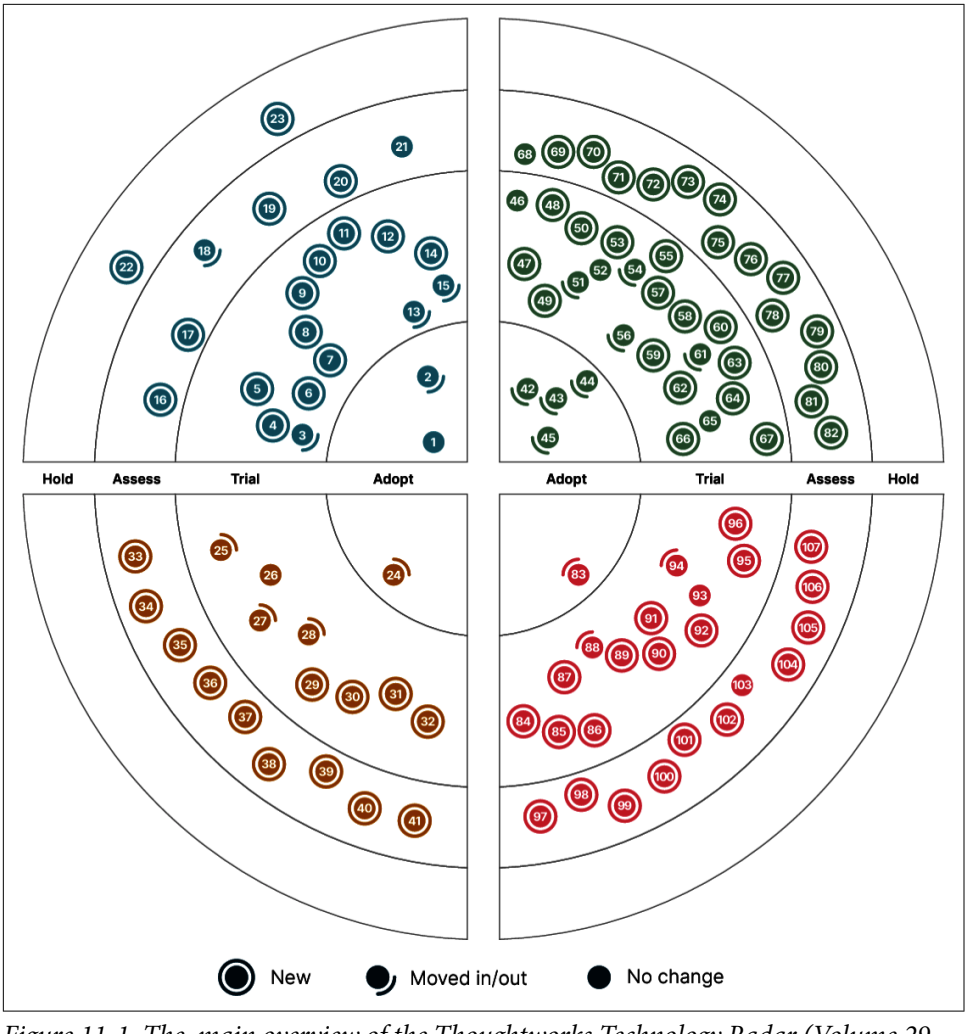


Figure 11-1. The main overview of the Thoughtworks Technology Radar (Volume 29, published September 2023)

How the Thoughtworks Technology Radar Works

Let's take a closer look at how the Thoughtworks Technology Radar works. You can see and interact with the latest version on the Thoughtworks website.

Figure 11-1 shows the September 2023 Thoughtworks Technology Radar overview. The radar consists of blips, which are then categorized into quadrants and rings. The quadrants represent:

- Techniques
- Tools
- Platforms
- Languages and frameworks

The Thoughtworks radar is interactive. Clicking on the Tools quadrant opens a zoomed-in view, with the names of each blip now listed and the names of the rings more evident. See Figure 11-2.

Each blip on the radar represents a technology or technique that plays a role in software. Their position in a given ring on the radar represents Thoughtworks's confidence in recommending that technology to their clients.

What are the rings? Starting at the outside and moving in, you have:

Hold
: Proceed with caution.

Assess
: Worth exploring with the goal of understanding how it will affect your enterprise.

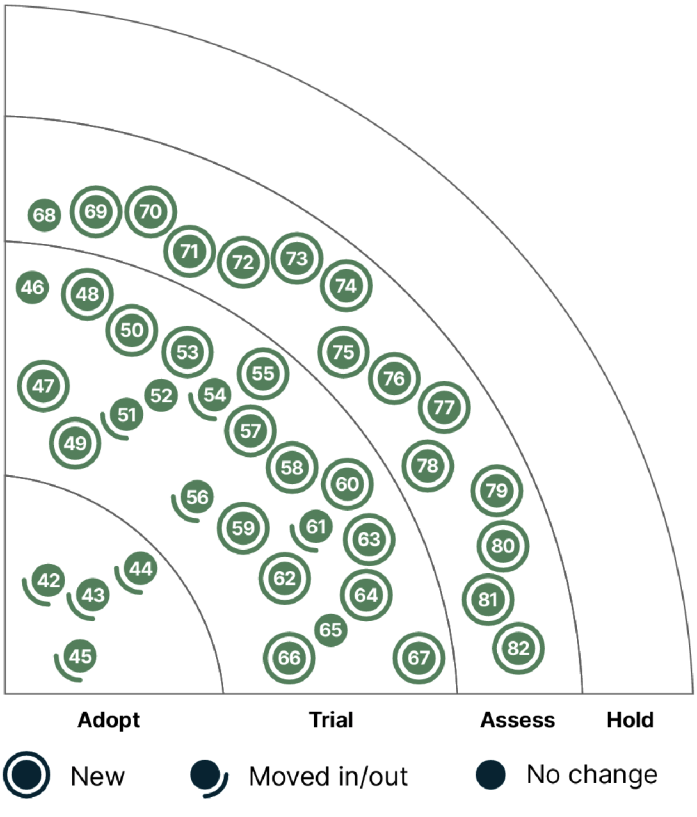
Trial
: Worth pursuing. It's important to understand how to build up this capability. Enterprises can try this technology on a project that can handle the risk.

Adopt
: Thoughtworks strongly feels that the industry should be adopting these items *where applicable*. Thoughtworks uses them *when it's appropriate* in their projects.

A blip combined with its ring position therefore clearly and succinctly captures Thoughtworks's current opinion on a topic.

As on a regular radar, blips are in motion. Their positions on the Thoughtworks Technology Radar change to represent Thoughtworks's increasing (or decreasing) confidence in recommending them as they move through the rings over time.

For example, the Mermaid blip is represented as a circle with a quarter ring around it, indicating it has been on previous radars and has moved in this particular edition. If you look really closely, you can see that on the Mermaid blip, the quarter ring is on the side closest to the center of the radar, which means its move was inward; Thoughtworks is increasingly confident in recommending it to clients.



43. Mermaid

Adopt

Mermaid lets you generate diagrams from a Markdown-like markup language. Since we last featured it in the Radar, Mermaid has added support for many more diagrams and integrations with source code repositories, IDEs and knowledge management tools. Notably, it's supported natively in popular source code repositories such as GitHub and GitLab, enabling the embedding of and easy updates to Mermaid diagrams in the middle of Markdown documentation. Many of our teams gravitate toward Mermaid as their diagram-as-code tool due to its ease of use, multitude of integrations and wide variety of supported diagram types that keep growing.

Figure 11-2. The tools quadrant of the Thoughtworks Technology Radar with the details of the Mermaid blip expanded (Volume 29, published September 2023)

Each blip is accompanied by a short piece of text that is special to each version of the radar. You can also view the blip's history to see how it has changed, or moved throughout the radar, over time. The history of Mermaid is shown in Figure 11-3.

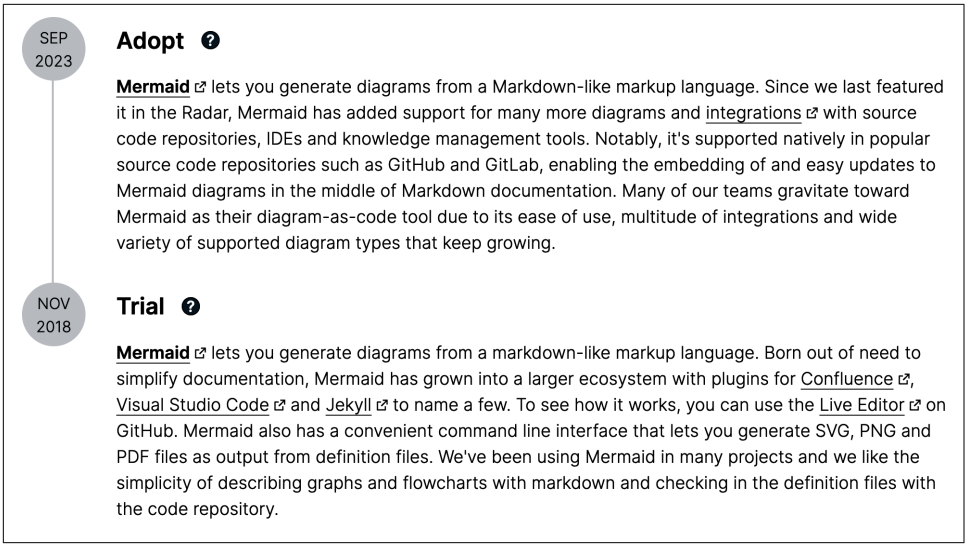


Figure 11-3. The full history of the Mermaid blip on the Thoughtworks Technology Radar

At the top of Mermaid's history page is the blip's current status ("Adopt") and when it moved to this status (September 2023). The text on the right gives Thoughtworks's current thinking on Mermaid as a tool and why the collective feels it deserves its place in the "Adopt" circle.

Complementing the blip's current position are its previous entries on the radar. For Mermaid, there is only one of these, from way back in November 2018 when it entered the radar in "Trial." You can see from that text what everyone thought about it then. This text is unedited from 2018.

Other blips have more entries because their progress (or otherwise) across the radar has been far more noteworthy and eventful. The OpenTelemetry blip is interesting. It has collected four entries since its first appearance in March 2017 but has never managed to move out of "Trial" status.

Other blips also have a long history, but their movement is not always toward the center. The AWS blip is illustrative here; it first appeared in July 2011 at "Adopt" sta-tus, with this ring position being reiterated in March 2012. However, in its third and final appearance on the radar in November 2018, AWS moved out to "Trial" because there were by then solid competitors in the market in the form of Microsoft Azure and GCP.

The AWS blip illustrates one final important point. While a blip might stay on the radar, it is not always represented visually. This is solely to preserve screen estate, highlighting what Thoughtworks thinks is most important at the time of the current version. All blips, current and future, can always be searched for.

It's Not Just Technologies that Belong on the Radar

Thoughtworks includes both patterns arising and antipatterns in the "Techniques" quadrant. These are a valuable place to share lessons and guidance that don't relate directly to a specific technology. I am always on the lookout for "ESBs in API Gateway's clothing", for instance.

Hopefully, you can see how rich a tool for capturing and sharing guidelines *over time* a technology radar is. Not only does it capture and organize a fantastic amount of information, but it also shows trends over time and (most important for our purposes) the tribal knowledge associated with historical movements and current positions of blips.

Your Internal Technology Radar Will Be Structured Differently

Although your technology radar will look and work similarly to Thoughtworks' Technology Radar,[2] your organization will make use of it slightly differently, both in terms of its content and its structure.

First, while Thoughtworks doesn't list any in-house tooling that their clients might have created (it's the clients' intellectual property, not that of Thoughtworks, after all), your internal radar is under no such restriction. Locating internal solutions in all quadrants, strategic or de facto, along with honest assessments of their applicability and current state as blips is a very powerful way to make it easy for everyone to know what's available to them in the form of invested-in strategic defaults. (I'll return to this in more detail in a later section of this chapter where I'll explain just how powerful this can be and how to harness it.)

Second, while your radar and the Thoughtworks version both capture and present guidelines about the adoption of current and past technology trends, *your context and focus will be very different*. The Thoughtworks radar is broadly focused: externally, across the entire technology sector. Your radar, on the other hand, will fit into a very specific niche: your decision processes and their encompassing sociotechnical system. While the Thoughtworks radar exists to help their clients make good decisions and avoid bad ones, your radar will focus on supporting distributed decision making within your specific internal landscape and climate.

What this typically means for the structure is that the quadrants stay the same (Techniques, Tools, Platforms, and Languages and Frameworks) but the rings reflect the

2 The biggest differences are that the backing site isn't included (I'll show how to make your own), and blip icons don't change to indicate movement between volumes.

transit of technologies through your organization (becoming "trial," "adopt," "hold," and finally "retire").

You're not bound to using these, but they are the ones I've repeatedly found most useful, and for the remainder of this chapter, they are the ones I'll use for examples.

Your Radar's Place in an Advice Process

How can an internal radar interact with your decision process? As with architectural principles, you can start by adding a section in your ADR template for Relevant Radar Blips as per the following example. Deciders can then list the pertinent blips in the ADR—both their position and detailed notes—which brings them explicitly into individual decisions.

EXAMPLE

ADR00X – [TITLE]

Date
Dec 22, 2022

Status
[DRAFT]/[PROPOSED]/[ACCEPTED]/[ADOPTED]/[SUPERSEDED]/[EXPIRED]

Decision
[A summary of your decision in a few lines.]

Relevant radar blips
[List here all blips that are directly relevant to this decision, whether the chosen option follows their guidance or takes another route. This can help create and focus both your options and their consequences.]

Context
[The context of your decision—the surrounding environment in which it was taken (including any constraints), the relevant steps which took you to this point, and the forces which made it necessary.]

A blip and its ring convey a lot on their own, but don't forget the explanatory text. This is a rich resource, especially if a blip history is also available. (I'll talk about how to capture blip history in "Updating Your Blips" on page 22.) For example, before trying something new—a tool, library, platform service, or technique—a team can see if anyone else in the organization has looked at it already, without having to ask. Deciders can see highlights of what others learned from the experience and more by referring to the blip's description text. If you choose to capture it in ADRs, this description

can flag which teams the experience comes from, which can point advice seekers in the right direction.

Consequently, the radar allows deciders to rapidly scan the range of pretried solution options at a glance. Furthermore, the ring position of blips illustrates clearly not only the current suggestions and associated guidance (relevant blips in "adopt") but also the investigated futures (relevant blips in "trial)" and remaining legacy (blips in "hold" and "retire"). Incorporating such guidance can improve your decisions significantly, embedding them more explicitly in your organization's current context.

The presence of relevant blips and their positions can feed directly into a decision ADR's options and their consequential pros and cons. If a blip is widely adopted, then making a decision that agrees with its position and description becomes moot. (This is a solid indication that you're not in the process of taking a "significant" decision.) However, as with architectural principles, a decision that looks like it is going to deviate from the current blip landscape—either by treating a blip as if it is in a different ring or by adding a new blip—must flag this fact explicitly in an ADR, explicitly addressing it in the text of the Context, Options, and Consequences sections. Even in these cases, the "default" (i.e., blip-aligned) options ought to be listed in the ADR, if only to articulate why they aren't preferred.

Highlighting relevant blips calls out any tension between an in-progress decision and the current collectively agreed technical guidelines as represented in the radar. This clarity is excellent grist for advice offering, whether it takes place in an architecture advice forum conversation or not. Any discussion, argumentation, and advice arising will likely focus on two key areas:

- Why does this decision need to take a different path?
- What is unsuitable about the defaults that are in existence?

Advice focused in this way is incredibly valuable. It can call in other blips, historical and up and coming, resulting in a rich and productive conversation. The resulting laser-targeted advice can then be recorded in the ADR and rolled into the decision.

However you use your radar to navigate your technical landscape, bear in mind that the radar's guidance (compared to principles) is advisory in nature rather than a reflection of a shared commitment. The radar is fundamentally a reflection of a current state and its past as well as the guidance arising from that. Radar blips give an idea of what, if anything, is the current de facto solution in a problem space: what's been done in the past and what teams might be experimenting with. Forging a new path when it comes to decisions that relate to blips is a lot less likely to raise eyebrows, but such a deviation should be specifically and concretely addressed in an ADR.

While the majority of radar-decision interactions flow from blips to decisions, this is not a one-way relationship. As I've just shown, decisions and the ADRs that result

from them can exert a force in the opposite direction, updating the radar. There are two main circumstances when this feedback could take place. I'll cover them in more detail later in the chapter, but I'll mention them now as they arise from the interaction between decisions and the radar.

First, while many deciders will flag their adherence to the current radar guidance by saying, "Our decision aligns to these blips," others will be saying, "Our decision contradicts the current blips and their position." Perhaps it's the spiking of an entirely new framework or a decision that moves a specific practice from "trial" to "adopt." In these cases, the radar may be due for an update.

For example, if a decision is taken to try out a new language, then a blip should be added in the "Languages and Frameworks" quadrant's "trial" ring. Or if a decision is taken to migrate a number of Kubernetes Pods from a previously standard homegrown kOps cluster running on AWS Elastic Container Service (ECS) instances onto Amazon EKS, then the EKS blip might move from "trial" to "adopt" while the kOps and ECS blips might move to "hold."

You could make updating the radar explicit via a process, though in my experience, this happens anyway without anyone having to push it explicitly. Remember, your goals here are the broadest engagement with your evolving architecture as possible, as well as a growing architectural mindset across all team members. I'll discuss how and when I go about this in "Updating Your Blips" on page 22.

Second, considering the longer-term blip lifecycle is useful for those wondering where to place strategic investments. Perhaps the time has come for the delivery platform team to make a self-serve template for a certain shape of Rust microservice, or on the other hand, time and effort perhaps ought to be put into the intentional removal of that last Bull mainframe running that old customer database. (I consider this in more detail in the "Consciously Managing Strategic iInterventions" section of a supplementary article on the book website.)

Creating Your Technology Radar

Let's now use the open source Thoughtworks "build your own radar" (BYOR) tool to build an interactive version of your radar. (You can find the example spreadsheet with links to the published radar here.) With that, you can visually map out the technology trends affecting your organization, past, present, and future. Importantly, alongside each of them you can capture the guidance that everyone wants to share about their relevant experiences.

To use the BYOR tool, go to the Build Your Own Radar page and follow the instructions to create a new spreadsheet for recording your blips. You're then ready to start with what can be the most fun part: collecting the blips. This is the equivalent of

sweeping a radar beam around a full 360 degrees, sensing what technologies and techniques are in each of your four quadrants.

After gathering the raw blip data from the experiences of your entire technology organization, you'll then go through the "sense-making" steps of sorting and validation, focusing, positioning, documenting, and finally publishing. All of these steps will happen in the spreadsheet. At any point in the process, you can pass the spreadsheet data to the BYOR tool to render it in the format I showed you in Figure 11-1.

Blip Gathering

In this step, the goal is to gather the raw, unfiltered suggestions for blips from all relevant parties. The task of gathering radar inputs and turning them into something that offers guidance to everyone is not a one-person job. Just as architectural principles should come from the collective, so should radar blips. Unlike architectural principles, however, blip gathering is best done asynchronously.

Your First Radar Sweep Takes the Longest

Your first sweep of your radar will take the longest because it is starting from nothing and capturing the most blips–everything you have now in your organization. It's establishing a baseline.

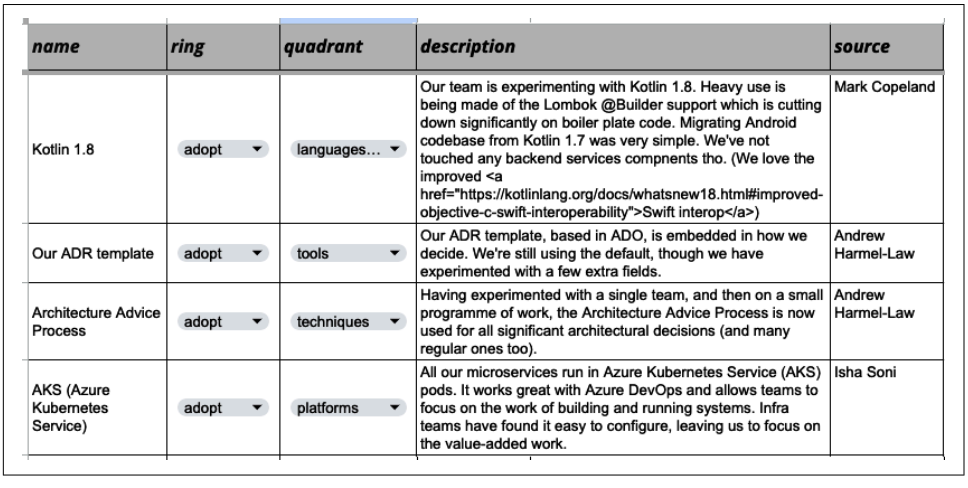
It's essential that you offer everyone the opportunity to be involved so that you get the most accurate and detailed overview of your current landscape and the prevailing climate. When you capture that, your radar stands the best chance of being useful in its task of offering guidance and aligning everyone.

To run a sweep, you first need to decide how much of your software engineering organization your sweep will cover. It works best to set this scope around one or more (or all) products *along their entire lifecycle*. Sweeps bounded by departmental, organizational, or role lines tend to miss important blips, such as testing tools or rollback techniques that will affect those same products.

This means you will be including disciplines like operations, UX, product, QA, security data, and delivery. Organization-supporting functions like people teams (aka human resources), finance, and legal are likely to fall outside your net, but you should give serious consideration to including teams like regulatory and others that, in other circumstances, are sources of CFRs. If you are in doubt, in my experience it's better to leave them out this time and include them in a future sweep or add their blips one at a time afterward.

Once you have your list of people, you can begin. The easiest way to do this is with a spreadsheet that has your columns and valid fields preset. Figure 11-4 shows what

a Google spreadsheet doing this might look like. (All the following spreadsheet images are linked to online versions.)



name	ring	quadrant	description	source
Kotlin 1.8	adopt	languages…	Our team is experimenting with Kotlin 1.8. Heavy use is being made of the Lombok @Builder support which is cutting down significantly on boiler plate code. Migrating Android codebase from Kotlin 1.7 was very simple. We've not touched any backend services compnents tho. (We love the improved Swift interop)	Mark Copeland
Our ADR template	adopt	tools	Our ADR template, based in ADO, is embedded in how we decide. We're still using the default, though we have experimented with a few extra fields.	Andrew Harmel-Law
Architecture Advice Process	adopt	techniques	Having experimented with a single team, and then on a small programme of work, the Architecture Advice Process is now used for all significant architectural decisions (and many regular ones too).	Andrew Harmel-Law
AKS (Azure Kubernetes Service)	adopt	platforms	All our microservices run in Azure Kubernetes Service (AKS) pods. It works great with Azure DevOps and allows teams to focus on the work of building and running systems. Infra teams have found it easy to configure, leaving us to focus on the value-added work.	Isha Soni

Figure 11-4. A Google spreadsheet used to gather blips containing some example data in each of the fields ("isNew" column hidden)

Notice that on the right, there is an extra column not in the Thoughtworks spreadsheet: *source*. I'd advise against adding fields apart from this one. As you create and start to use the radar, it helps to know who submitted a blip, whether an individual or a team. This allows everyone to go back to the submitter if something about the blip is unclear. And it has benefits when the radar is in use, but I'll get to those later.

When you share the blip collection sheet with everyone, make it clear what your intentions are by including a cover sheet that describes the technology radar concept, explains the process you are asking everyone to follow, and links to the Thoughtworks Technology Radar and explainer videos, so contributors have a good idea of what they are doing. It also makes sense to state clearly how this will fit into your decision process.

How you drive the collection will depend on your organization's engineering culture. I've worked with clients who've announced the call for blip submissions at an all-hands meeting and then shared the empty spreadsheet with everyone with a deadline for completion; everything else just self-organizes. I've also had clients who began similar announcements but then hosted a series of facilitated sessions, moving team by team and collecting blips that way. Most organizations sit somewhere between these two.

The description is the most valuable field to capture at this stage. Encourage contributors to dump raw information in here. If there is disagreement about a blip, capture that too, and if different teams have different thoughts on the same blip, suggest that they log their versions separately. You're very much not looking for agreement here,

simply awareness. Commenting on the descriptions of blips that others added should be encouraged. If disagreement exists, then surface it.

Letting teams and other groups self-organize works well, but be aware of the specifics of your culture because you want to be confident that the blips gathered will be broadly representative and of a solid enough quality to take to the next stage. By this, I mean:

- For each blip, all columns need to be filled in.
- This is an initial scan: you want *everything*, future, current, and legacy. If it's affecting your organization right now, capture it.
- It's not just technology—remember techniques. Don't restrict yourself to industry-standard approaches either. Capture how *you* do it, pattern or antipattern.
- Don't worry about duplicates—the "focusing and positioning" stage resolves that.
- Capture all the major versions of all the things you use. If you have Java Runtime Environments (JREs) from version 1.1 all the way to version 21, then add blips for each one. If you have several versions of a library, list them all out, too.[3]
- Capture all the variety of the things. If you mix home-developed Kubernetes clones, kOps, EKS, and a million other flavors, list them all out.
- Think about the entire software delivery lifecycle, from ideation to sunset.
- Think about the entire tech landscape, from Internet of Things (IoT) to offline cold storage in a permafrosted Icelandic bunker.
- Don't forget standards: open, closed, and de facto internal.

Lock Then Copy the Sheet After Every Step Is Completed

Before moving on to each subsequent step in radar creation, lock your blip-gathering sheet ("protect" is what Google Sheets calls it), then take a snapshot copy to work on. This means the data won't continue to change underneath you as latecomers try to add their last few blips, but it will mean they are disappointed. To avoid this, set a deadline for the current stage.

Blip Sorting and Validation

At this point, you will have a lot of blips in their raw form, shown in Figure 11-5. If you rendered your radar right now, it'd be unfocused and not very usable. More

3 This is advice, not a rule. Not everyone follows semantic-versioning principles, especially when marketing gets involved. The goal is that you get enough details of blips that are significantly different enough to be treated separately.

important, it stands little chance of representing a *balanced and nuanced* view of your technology landscape and climate. To make sense of this raw data, the collective will need to be involved a second time to focus and position the collected blips. For that activity to be a success, whoever is running the radar-creation process will need to do some prep to first sort and validate the blips. That is the goal of this step.

name	ring	quadrant	description	source
Kotlin 1.5	adopt	languages…	First version of Kotlin at NuOrg. Used for a few experiments which went well, ending up in prod. Some backend services. None mission critical.	
Kotlin 1.8.10	trial	languages…	We've started playing with kotlin again in our team.	Seema Satish
Swift 4	adopt	languages…	First version of Swift that we used. Adopted completely after we dropped ObjectiveC. (Hiring Swift developers was easier.)	Mark Copeland
Azure Functions	trial	platforms	We used them to build the entire Customer Onboarding microsite. This is a great use case as it runs infrequently, and when it does it is fundamentally a complicated, async data ingestion pipeline.	Isha Soni
kotlin 1.7	adopt	languages…	Kotlin version our Android app currently targets.	Mark Copeland
Kotlin 1.8			Our team is experimenting with Kotlin 1.8. Heavy use is being made of the Lombok @Builder support which is cutting down significantly on boiler plate code. Migrating Android codebase from Kotlin 1.7 was very simple. We've not touched any backend services compnents tho. (We love the improved Swift interop)	Mark Copeland
Our ADR template	adopt	tools	Our ADR template, based in ADO, is embedded in how we decide. We're still using the default, though we have experimented with a few extra fields.	Andrew Harmel-Law
Architecture Advice Process	adopt	techniques	Having experimented with a single team, and then on a small programme of work, the Architecture Advice Process is now used for all significant architectural decisions (and many regular ones too).	Andrew Harmel-Law

Figure 11-5. Selection of raw, unsorted blip data after gathering but before sorting and filtering, focusing, and positioning ("isNew" column hidden)

This preparation involves tidying up the blips in a way that respects the raw data you have at the moment.[4] Your radar should reflect what is going on in your organization *right now*, not where the organization hopes to be. Bear this in mind as you sort and validate each blip in turn, undertaking the following tasks in the following order:

4 A bit like tidying your child's bedroom. Is that a random pile making its way glacially toward the bin or is it a carefully thought-through and optimized filing system? Best to leave it for now and check with them first…

1. Ensure that all the blips have isNew set to “true.”
2. Correct omissions (i.e., put things in quadrants or rings if these were not set).
3. Clarify with the author if a blip is not clear.
4. Group rows that are the same.
5. Group rows that are related.
6. Group rows by quadrant.
7. Group rows by proposed ring.

Postsorting and postfiltering, you should have something like the sample in Figure 11-6. Every blip is set now to “isNew=true” as this is the first radar, the source has been validated, and the ring and quadrant have been guessed for Kotlin 1.8, which is now next to the blip for Kotlin 1.8.10. The teams of the blip submitters have also been added for clarity.

name	*ring*	*quadrant*	*isNew*	*description*	*source*
Kotlin 1.5	adopt	languages…	TRUE	First version of Kotlin at NuOrg. Used for a few experiments which went well, ending up in prod. Some backend services. None mission critical.	Paul Fitzsimons (Payments Back End)
Kotlin 1.7	adopt	languages…	TRUE	Kotlin version our Android app currently targets.	Mark Copeland (Mobile)
Kotlin 1.8	adopt	languages…	TRUE	Mobile team is experimenting with Kotlin 1.8. Heavy use is being made of the Lombok @Builder support which is cutting down significantly on boiler plate code. Migrating Android codebase from Kotlin 1.7 was very simple. We've not touched any backend services compnents tho. (We love the improved Swift interop)	Mark Copeland (Mobile)
Kotlin 1.8.10	trial	languages…	TRUE	We've started playing with kotlin again in our team.	Seema Satish (Infra)
Swift 4	adopt	languages…	TRUE	First version of Swift that we used. Adopted completely after we dropped ObjectiveC. (Hiring Swift developers was easier.)	Mark Copeland (Mobile)
Azure Functions	trial	platforms	TRUE	We used them to build the entire Customer Onboarding microsite. This is a great use case as it runs infrequently, and when it does it is fundamentally a complicated, async data ingestion pipeline.	Isha Soni (Cards Back End)
AKS (Azure Kubernetes Service)	adopt	platforms	TRUE	All our microservices run in Azure Kubernetes Service (AKS) pods. It works great with Azure DevOps and allows teams to focus on the work of building and running systems. Infra teams have found it easy to configure, leaving us to focus on the value-added work.	Isha Soni (Cards Back End)
Architecture Advice Process	adopt	techniques	TRUE	Having experimented with a single team, and then on a small programme of work, the Architecture Advice Process is now used for all significant architectural decisions (and many regular ones too).	Andrew Harmel-Law (Architecture)

Figure 11-6. Selection of sorted and filtered blip data, ready for focusing and positioning

Respect the Submitted Raw Blip Data

It's very important to respect the inputs that have been provided so far.[5] By all means, correct spelling mistakes in the data entry and group them to make the upcoming focusing session efficient, but don't stray beyond this list—by deduplicating, for example. Just because you think it's a double or disagree with it doesn't mean you can change it. That kind of activity comes next.

Blip Focusing

After sorting and validating the blip information, it's time to involve the collective again to further focus them. In this step, the goal is to check that the blips are valid, verify the blips' description text, and decide which quadrant the blips belong in. Unlike the gathering step, the focusing step should be done synchronously in a workshop. These workshops can often take up to four hours.

The attendees for this blip-focusing session will ideally be the same group who were asked to contribute the raw blips in the first place. However, if this is an unmanageable number of people to have tied up in long working sessions, ask each group to send one or more delegates whom everyone in the group empowers to act on their behalf. By *group,* I mean a development team or one of the disciplines (e.g., UX, product, infra, InfoSec, etc.).

Even with reduced numbers, blip focusing is still one of those "expensive" workshops if you multiply the number of attendees by the number of hours it takes, which is why the sorting and validation step is so important, as you want to be able to move through similar blips rapidly.

Don't Scrimp on the Focusing Time

If you're sweeping your radar cross-organization, then gathering the number of invitees for a four-hour session might simply be impractical. Splitting the session into two 2-hour pieces or even four slots of one hour each is fine. It's also workable to run sessions for different products. For example, gather the teams working on a cloud-based SaaS solution (not just the development teams—don't forget the platform and supporting functions) for one session, and then separately the teams working on the mobile app or some on-prem, thick-client systems.

5 This is the "Monira Rhaimi method." Monira would never move sticky notes that had been placed by others if they weren't present. The notes represented their authors' conception of something, which should be treated with respect if we expect them to trust us.

Despite all these warnings, blip focusing can be great fun. There are very few opportunities for everyone to get together and discuss the technology trends that are affecting your organization right now.

During the session (or sessions, if you split them up), work through every blip gathered, one after another. This is why it helps to have everything sorted in advance. If there are a lot of rows that cover the same thing, then treat them all together.

For each blip, check if it is valid, clarifying the text in all cells as appropriate. If it is a duplicate, then discuss and merge, taking care not to lose any of the valuable description data and sources in each of the individual blips. If someone remembers something that everyone forgot, then add it to an existing blip or a new one.

Individual Blips for Major Versions of the Same Technology Trend

If you are running multiple major versions of something, be that a framework or library, a language, or even a platform, represent each version on the radar. This allows you to show where the collective thinks everyone should be moving as a default and where all your legacy is. This is one area where your organization's radar will differ from the Thoughtworks one.

It is also important to agree in which ring a blip sits. It will be self-evident in most circumstances, but when it's not, you can refer to the ring definitions and if you need to, split a blip in two so that it can sit in both places. If this happens, make sure that the names and descriptions of the new blips refer to the reason they are in that specific ring. (Doing this often clarifies each of the blips and fits them to their ring homes.) For example, you might have "Azure Functions for infrequent workloads" in "adopt" and "Azure Functions for flexible workloads" in "trial."

As the discussion progresses, make sure you (or another nominee) capture all the commentary that surfaces. Disagreement is fine; you're not looking for consensus. If there are two ways to do things, show it. Two conflicting opinions as to the best way to do or use something? Make both clear.

You will likely find that the "techniques" quadrant starts off relatively empty compared to the others, but watch out as discussion continues for things that could go in here. This is a place where tribal memory and lessons learned can be captured. The Thoughtworks radar has some great examples of this, such as "tracking health over debt", "bounded buy", "Lambda pinball", "team cognitive load", "miscellaneous platform teams", and "ESBs in API Gateway's clothing". (As you can see, this is an opportunity to get creative with names. This isn't just for fun. If a name is memorable, then a technique stands a better chance of being adopted or avoided if it's a bad smell.)

Once you're done, you should have a selection of blips, something like the sample in Figure 11-7. Note that Kotlin 1.8.10 has been merged into the Kotlin 1.8 blip. Also, some more detail on other teams' use of Azure Functions has been added.

name	ring	quadrant	description	source
Kotlin 1.5	adopt	languages…	First version of Kotlin at NuOrg. Used for a few experiments which went well, ending up in prod. Some backend services. None mission critical.	Paul Fitzsimons (Payments Back End)
Kotlin 1.7	adopt	languages…	Kotlin version our Android app currently targets.	Mark Copeland (Mobile)
Kotlin 1.8	adopt	languages…	Mobile team is experimenting with Kotlin 1.8. Heavy use is being made of the Lombok @Builder support which is cutting down significantly on boiler plate code. Migrating Android codebase from Kotlin 1.7 was very simple. We've not touched any backend services compnents tho. (We love the improved Swift interop) We've started playing with Kotlin again in our team (Infra Team)	Mark Copeland (Mobile), Seema Satish (Infra)
Swift 4	adopt	languages…	First version of Swift that we used. Adopted completely after we dropped ObjectiveC. (Hiring Swift developers was easier.)	Mark Copeland (Mobile)
Azure Functions	trial	platforms	A few teams have experimented with Azure Functions. All but one have used them for back-end housekeeping. Cards team used them to build the entire Customer Onboarding microsite. This is a great use case as it runs infrequently, and when it does it is fundamentally a complicated, async data ingestion pipeline.	Isha Soni (Cards Back End), Seema Satish (Infra)
AKS (Azure Kubernetes Service)	adopt	platforms	All our microservices run in Azure Kubernetes Service (AKS) pods. It works great with Azure DevOps and allows teams to focus on the work of building and running systems. Infra teams have found it easy to configure, leaving us to focus on the value-added work.	Isha Soni (Cards Back End)
Architecture Advice Process	adopt	techniques	Having experimented with a single team, and then on a small programme of work, the Architecture Advice Process is now used for all significant architectural decisions (and many regular ones too).	Andrew Harmel-Law (Architecture)

Figure 11-7. Selection of focused blips ("isNew" column hidden)

Blip Positioning

Once you have all your blips deduped and allocated to the right quadrants, you can collectively position them, assigning each blip to the appropriate ring. If you are keeping the defaults, these will be trial, adopt, hold, or retire. Unlike focusing, this can be done offline and asynchronously, which also allows everyone in the greater blip-providing community to contribute. Invite everyone who could have submitted a blip, whether they chose to or not.

Prepare for positioning by taking a copy of the sheet with the focused blips on it and making a new column for each person who will vote. Each cell in these new columns should only allow each of the rings and "not voted" as values. Open the voting by sharing the blip-positioning sheet with everyone who is enfranchised. Each voter then votes for the ring they think each blip should be placed in.

Voting Might Flush Out More Comments—Welcome This!

There's nothing like a radar that is about to make its way into the open to flush out final comments on the blips. To capture these, encourage everyone who is voting to add these closing thoughts in the form of comments on the relevant blip's description cell. That way, voices can still be heard, and the last bit of context juice can be squeezed from everyone.

As with blip gathering, set a time window for this voting (a week is generally good, though if engagement is high, then one to three days can work, too) and remind people when the voting window is about to close. Once the voting window closes, protect the sheet so that no more voting can take place. The position of each blip is the ring with the majority of votes. Crunch the results and set the ring position for each blip accordingly, as shown in Figure 11-8. You can see that while they are not always unani-mous, the votes all have clear winners. Some blips have shifted position: Kotlin 1.5 has moved to "retire," and Kotlin 1.8 has moved to "trial."

name	ring	isNew	Votes: Paul	Votes: Seema	Votes: Mark	Votes: Isha	Votes: Andrew
Kotlin 1.5	retire	TRUE	retire	hold	retire	retire	retire
Kotlin 1.7	adopt	TRUE	adopt	adopt	adopt	adopt	adopt
Kotlin 1.8	trial	TRUE	trial	adopt	adopt	trial	trial
Swift 4	adopt	TRUE	adopt	adopt	adopt	adopt	adopt
Azure Functions	trial	TRUE	trial	adopt	trial	trial	trial
AKS (Azure Kubernetes Service)	adopt	TRUE	adopt	adopt	adopt	hold	adopt
Architecture Advice Process	adopt	TRUE	adopt	adopt	adopt	adopt	adopt
Swift 5	trial	TRUE	trial	trial	trial	trial	trial
iOS 12 to 14	retire	TRUE	retire	retire	retire	retire	retire

Figure 11-8. Selection of blips and their votes ("description," "quadrant," and "source" columns hidden)

You are now ready to move to the final stage of preparation prior to publishing.

Blip Documenting

At this point, you have all the raw information required to create your first technology radar, but it's not yet in a digestible form. The goal of this step is to take all the raw information from the descriptions, as well as any further comments that arose during the focusing and positioning steps, and turn that into a short paragraph or two of useful prose for each blip that would support the organization's decision process.

While refining the documentation for each blip, it is important to retain all the information gathered from the previous steps. Capture the points of tension or conflict as well, as every bit of information is useful. You may also want to refer (and link) to other blips or to other architecture documentation, such as your tech strategy, high-level architecture, ADRs, and more.

At Thoughtworks, a few individuals take on this job, but each blip description is shared internally for everyone's comment and feedback via a universal, editable Google document. This allows everyone to see and comment on the proposed text for each blip. This text should capture how the blip is viewed by the collective, any points of tension or conflict that have been identified during the latest sweep, and any other information that may assist someone when making a decision that pertains to the blip.

Again, this commenting/feedback phase has a time box placed on it, after which the finalized text will be copied back into the spreadsheet ready for publishing, which is the final step.

Radar Publishing

Once the blip descriptions have been written and reviewed, you are ready to publish the first edition of your radar.

When you use the BYOR tool, there are a number of ways to host the result. The simplest, which involves no infrastructure on your end, is to use the free Thoughtworks hosting service. This has a few restrictions. You have to use a Google spreadsheet as your data source, and unless you choose to protect it, your radar effectively becomes public. If you are using this approach, you just need to provide it with a link to your private or public Google spreadsheet of blips. (Don't forget to protect it so no one can edit it.) This will then render the contents of your spreadsheet just like the Thoughtworks one, allowing everyone to interact with it in the same way.

If this hosting method or the associated Thoughtworks terms of use contradict your organization's InfoSec and data security policies, you can still use it, but you will need to find a different place to run it; it's just a JavaScript npm service reading from URL-accessible files, and a Dockerized version is available. Alternatively, there is a plug-in for Spotify's Backstage developer portal, and other alternatives might be available by the time you read this.

However you host your technology radar, when it is published, it will look like the Thoughtworks one. Figure 11-9 shows our example radar published with the Kotlin 1.8 blip highlighted and the tidied-up description on the right. Note that all the blips will display as "new" as this is the first edition of your radar.

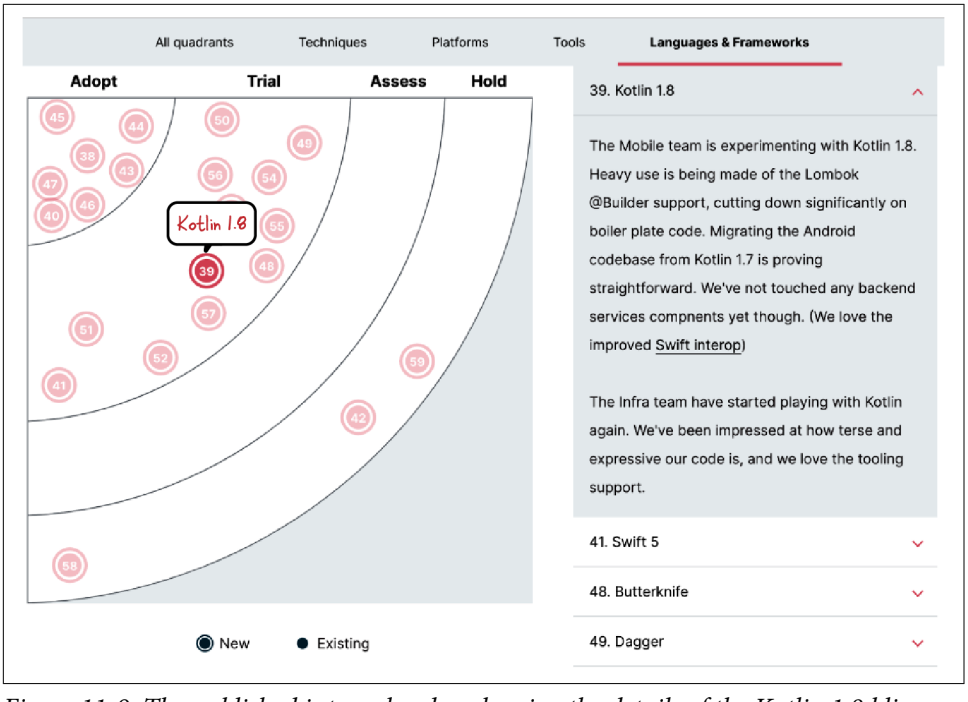


Figure 11-9. The published internal radar, showing the details of the Kotlin 1.8 blip

Once your technology radar is published, you can send out an announcement email with a link to your interactive radar so that everyone can start using it in their decisions. You'll also want to remind everyone through any other communication channels where architecture decisions are discussed, such as an architecture advice forum. Make it clear that while your radar is in the same format as the Thoughtworks one, it serves a different purpose: to capture the collective guidance in a format that can support decentralized deciding.

Don't forget to take advantage of the fact that you can generate a PDF of your radar. This is a great alternative for folks who prefer to consume their information in a static format. You can also use it as a snapshot of the current state of the radar that can be archived.

However you choose to publish the radar (on the Thoughtworks hosting service or on your own), or however your people choose to consume the information (as an interactive radar, a text-only document, or a PDF), the goal is to socialize your radar so that it becomes another useful tool supporting organizational alignment and your decision process as you collectively practice software architecture.

Share Your Radar with New Joiners

Share the radar with your new hires so that they can familiarize themselves with your current technology and technique landscape as well as its history and the associated guidelines that the collective have provided.

Updating Your Blips

One tech-radar sweep isn't enough. A traditional radar used to track aircraft sweeps regularly and continuously, capturing updated views of the surrounding airspace. Similarly, a technology radar should periodically sweep the current tech landscape and organizational climate, turning them into updated guidelines.

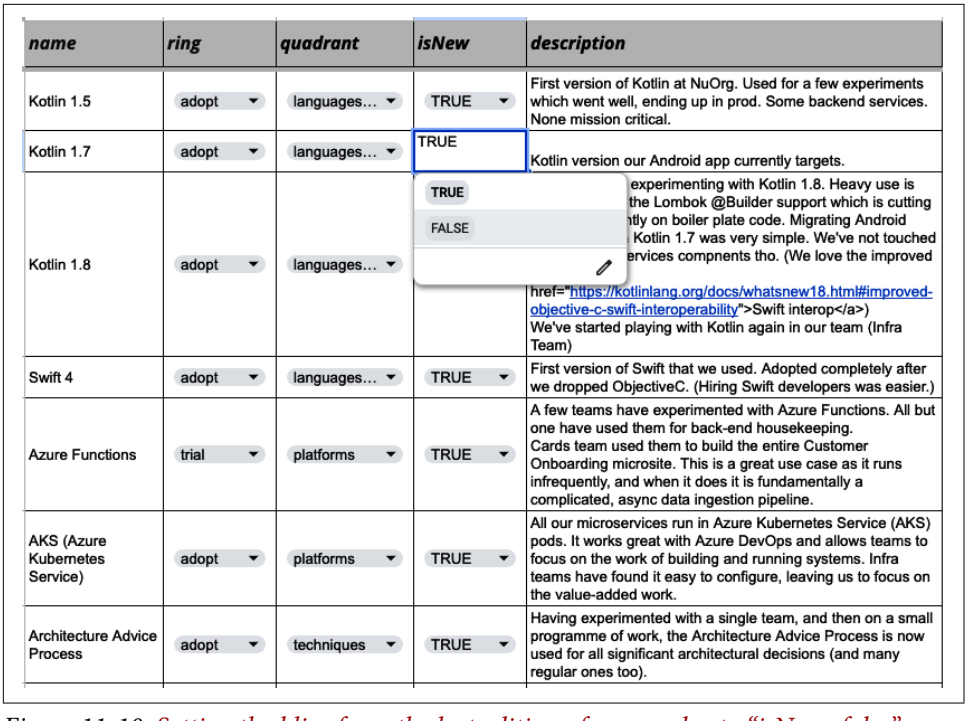
The frequency of these sweeps depends mainly on how rapidly these environmental factors are changing. I've seen quarterly cadences work, and half-yearly, too. The key is to pay attention to how the radar is being consumed (or not) at the advice forum and elsewhere. That should give you a good idea for when it's worth investing in a refresh.

Periodic Resweeps

Always plan for periodic resweeps, whether or not you capture ad hoc updates arising from individual decisions. These follow a very similar set of steps as the initial creation of the radar, but with a set of preexisting blips. The second and subsequent gatherings will go much more quickly because they harvest far fewer blips.

Prepare for a resweep by copying the published blip data into a new sheet and setting all the previous blips' "isNew" values to "false" (Figure 11-10). You can then encourage the blip submitters to contribute to this new sheet, once again setting a deadline for blip submissions. (Remember to set the previously published radar spreadsheet to read only and label it clearly.)

Make it clear to contributors that for the existing blips, this is not the time to suggest changes in ring position, but additional information in the description cells is welcome. This should be appended to what is already there, flagging who contributed it. This might be as simple as recording that more teams have adopted a blip or that they have moved away from it.



name	ring	quadrant	isNew	description
Kotlin 1.5	adopt	languages…	TRUE	First version of Kotlin at NuOrg. Used for a few experiments which went well, ending up in prod. Some backend services. None mission critical.
Kotlin 1.7	adopt	languages…	TRUE	Kotlin version our Android app currently targets.
Kotlin 1.8	adopt	languages…	TRUE / FALSE	experimenting with Kotlin 1.8. Heavy use is the Lombok @Builder support which is cutting tly on boiler plate code. Migrating Android Kotlin 1.7 was very simple. We've not touched ervices compnents tho. (We love the improved href="https://kotlinlang.org/docs/whatsnew18.html#improved-objective-c-swift-interoperability">Swift interop) We've started playing with Kotlin again in our team (Infra Team)
Swift 4	adopt	languages…	TRUE	First version of Swift that we used. Adopted completely after we dropped ObjectiveC. (Hiring Swift developers was easier.)
Azure Functions	trial	platforms	TRUE	A few teams have experimented with Azure Functions. All but one have used them for back-end housekeeping. Cards team used them to build the entire Customer Onboarding microsite. This is a great use case as it runs infrequently, and when it does it is fundamentally a complicated, async data ingestion pipeline.
AKS (Azure Kubernetes Service)	adopt	platforms	TRUE	All our microservices run in Azure Kubernetes Service (AKS) pods. It works great with Azure DevOps and allows teams to focus on the work of building and running systems. Infra teams have found it easy to configure, leaving us to focus on the value-added work.
Architecture Advice Process	adopt	techniques	TRUE	Having experimented with a single team, and then on a small programme of work, the Architecture Advice Process is now used for all significant architectural decisions (and many regular ones too).

Figure 11-10. Setting the blips from the last edition of your radar to "isNew=false" ("source" column hidden)

Newly submitted blips will most likely appear first in the "trial" ring. It is also likely that a greater number of techniques will be submitted for this quadrant than when the first scan took place. This seems to happen because the concept of this quadrant tends to sink in only after the initial collection, but this section really blossoms as organizational learning kicks in. Blips in the "techniques" quadrant are an excellent place to make guidance born of experience actionable. Blips can reaffirm what has been learned, making it available to those who have yet to join or have yet to take on the responsibility of deciding. Some classic examples of this from the Thoughtworks radar include: "SAFe™", the aforementioned "ESBs in API Gateway's clothing", and "Cloud lift and shift".

Wherever new blips land, you will again have to perform the same tidying-up activities as in "Blip Sorting and Validation" on page 13 to prepare for the next collaborative stages. The process of sorting and validating the new and updated blips is exactly the same as the original process.

The subsequent stages also proceed in a similar fashion to the first pass, but less time needs to be allocated. Sorting and validation and focusing need take place only for new blips, but the former is still best achieved in a workshop, which will take significantly less time than the first time around. Two hours is usually appropriate, but you will know how long you need based on your experience from the first workshop.

(Re)positioning applies to all the blips and should again be done by opening up the sheet to all in the blip-providing community. Not every blip needs to move, however; your radar should reflect reality, and if nothing has changed, then there is no need to update anything.

After (re)positioning is complete, documentation needs to reconsider all blips, too (though not all preexisting blips may need an update if they've not moved rings). This is a great time to discuss and provide updated guideline notes for all blips that stay on the radar, even if this text is largely based on the version from the previous edition's blip. Calling out that something about a blip hasn't changed can also be useful for the reader. Remember that the goal is to make your collective wisdom and experience explicit, whatever that is. On the other hand, if a blip has moved, then it's a great opportunity to explain why in the text. If increased—or decreased—numbers of teams are using it, then this should be reflected. Finally, if a blip is no longer applicable, then it should be removed completely. (I'll discuss strategic divestment and retiring legacy technologies in the final section of this chapter.)

Resweeps update your radar, but they have another benefit. By retaining the history of blips, they provide a valuable learning resource and reference library. But wholescale resweeps aren't the only way to capture blip updates.

Ad Hoc Updates Based on Shared Experience

While periodic resweeps keep your radar entirely up to date, they're not the only source of changes. Blip additions, movements, and clarifications should also emerge from the advice process and architectural decisions.

Recall that a decision that effectively adds a new blip (either because there are currently no blips that apply in that circumstance or because all those that do apply aren't sufficient or satisfactory for some reason) or a decision that advocates for a blip to be treated as if it were in a different ring is a solid indication that a decision is "significant." This fact should be highlighted in the ADR and advice sought. But what happens after the decision is taken and the ADR adopted? This should (at least) flag the blip as a candidate for an ad hoc update of the technology radar.

Decision-driven blip additions or updates are best moved forward via another ADR. If it's an entirely new blip, then this ADR should be raised by the team or individual who took the initial decision. If it's a blip update, then this responsibility will likely fall to someone with a cross-system (probably architecture, but possibly a platform team) role.

ADRs are a great format for such changes, and they can be very lightweight. Unlike with updating the set of principles, it is sufficient to add or update the principle in reference to the ADR that prompted the change, sharing the proposed new ring, the new text, and the reason for the addition or move.

If you choose not to update the radar at this time, don't worry. You can always catch the blip and update it when your next sweep comes around, and you have your explicit flagging of the deviation in the ADR. Whenever I postpone a radar update, I keep a list of these potential ADRs, which I can use to prompt everyone when the next sweep takes place to ensure that nothing gets missed. These can also be checked when another decision comes along that looks like it's proposing that the same blip be moved. (It needn't be the same type of change—remember, ADRs are great for transparent technical discussions. When combined with an advice process, they have a high tolerance for a diversity of voices and advice.)

Capturing Previous Blips and Their History

However you update blips, it can be valuable to keep track of how they have changed over time. If you choose to do this, you can create something that works similarly to the history pages of the Thoughtworks Technology Radar.

To create blip histories, it's best to have a wiki-type environment where each blip gets its own page. This can also help when you retire blips from the main radar but want to preserve the fact that they were once blipped.

For each blip in the original radar, move it to your wiki, one blip per page. Group these pages together based on the quadrants if you like. (It makes browsing easier.) When you create a new page, follow the structure and layout of the Thoughtworks Technology Radar. When a blip makes it to a subsequent edition of the radar, put the new ring and details at the top of the page.

Link the blips in the current radar to their history wiki pages by adding a hyperlink to the blip description in the spreadsheet. This allows viewers of the radar to click through to these history pages. Having this guidance at your fingertips helps greatly when making new decisions.

Further Uses for a Technology Radar

As a technology radar becomes embedded within your advice process and wider sociotechnical ecosystem, you may realize that it offers extra opportunities. I've collected the opportunities I see most frequently into a separate article that covers additional blip information you may want to capture as well as how to use your radar to consciously manage the higher levels of strategic interventions that I introduced in Chapter 9, both investments and divestments.

Conclusion

This chapter described a fourth supporting element: your own technology radar. I considered how it offers a fun way to both sense for and make sense of the more subtle signals in your sociotechnical ecosystem and offer contextualized advice on them. I showed how important the end result is as a support for the advice process generally as well as a source of potential strategic investments and a way to manage strategic divestments.

The technology radar takes the last place in my go-to mechanisms for supporting and promoting decentralized and feedback-oriented practices of software architecture. There might be alternatives—especially the supporting elements—and I encourage you to experiment with them all, making sure you keep the goals of your approach in mind.

As you probe, you will learn a lot more about the practice of decision making and its place in both software development and sociotechnical systems. Part 3 focuses on both topics from multiple angles, with the goal of enabling everyone who decides to practice decentralized architecture for flow and feedback.

About the Author

Andrew Harmel-Law is a tech principal at Thoughtworks, specializing in domain-driven design, org design, software and systems architecture, Agile delivery, build tools, and automation. Andrew's experience spans the software development lifecycle and many sectors.

Andrew is also an author and trainer for O'Reilly, having written not only this book about facilitating software architecture but also a chapter about implementing the Accelerate/DORA four key metrics. Andrew also runs regular online training sessions such as "Domain-Drive Design (First Steps)" and "Architecture Decision Making by Example." What motivates Andrew is the humane delivery and sustainable evolution of large-scale software solutions that fulfill complex user needs. Andrew understands that people, architecture, process, and tooling all have key roles to play in achieving this. Andrew has a great passion for open source software and its communities. Andrew has been involved with OSS to a greater or lesser extent since their career began; as a user, contributor, expert group member, or paid advocate—most notably as one of the Jenkins JobDSL originators.

Andrew enjoys sharing their experience as much as possible. This sharing is seen in not only formal consulting engagements but also informally through mentoring, blog posts, conferences (keynoting, speaking, and organizing), and open sourcing their code.

If you want to reach out to Andrew, you can try LinkedIn, Mastodon, Bluesky, or (if you must) Twitter.

Colophon

The animals on the cover of *Facilitating Software Architecture* are two-barred crossbills (*Loxia leucoptera*). A North American version—currently listed as the same species—is known as the white-winged crossbill (two-barred crossbills tend to be a bit larger than their North American counterparts).

Two-barred crossbills inhabit the Palearctic, mostly in Russia. Their preferred habitats involve spruce trees and cones, which are their source of food. They use their unusual beak shapes to open the cones. Fun fact: two-barred crossbills are three times more likely to have lower beaks crossing to the right than to the left.

The male plumage is red while the female is yellow-green. The female typically builds the nest, lays three to four eggs, and incubates them. After the eggs hatch, the chicks are fed by both parents until they fledge 22 to 24 days later; they then remain with their parents for up to six weeks.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on an antique line engraving from *British Birds*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.